Journal of Computer Applications ISSN 1001-9081 2017
计算机应用,2017 CODEN JYIIDU http://www.joca.cn

最新录用 doi

基于 Greenplum 数据库的查询优化

邹承明[1,2]，谢义[1,2]，吴佩[1,2]

(1.交通物联网技术湖北省重点实验室， 武汉 430070；
2.武汉理工大学 计算机科学与技术学院， 武汉 430070)

(*通信作者电子邮箱 17607197069@163.com)

摘 要: 针对分布式数据库查询效率随着数据规模的增大而降低的问题，以 Greenplum 分布式数据库为研究对象，从优化查询路径的角度提出一个基于代价的最优查询计划生成方法。首先，该方法设计一种有效的代价模型来估算查询代价；然后，采用并行最大最小蚁群算法来搜索具有最小查询代价的连接顺序，即最优连接顺序；最后，根据 Greenplum 数据库对查询计划中的不同操作的默认最优选择，得到最优查询计划。采用该方法在自主生成的数据集与 TPC-H（事务处理性能理事会测试基准）的标准数据集上进行了多组实验。实验结果表明，所提出的优化方法能有效地搜索出最优解，获得最优的查询计划，从而提升 Greenplum 数据库的查询效率。

关键词: 分布式数据库；Greenplum 数据库；最优查询计划；代价模型；最优连接顺序

中图分类号:TP311 **文献标志码**：**A**

Query optimization based on Greenplum database

ZOU Chengming[1,2], XIE Yi[1,2], WU Pei[1,2]

(1.Hubei Key Laboratory of Transportation Internet of Things, Wuhan 430070, China;
2.School of Computer Science and Technology, Wuhan University of Technology, Wuhan 430070, China)

Abstract: In order to solve the problem that the query efficiency of distributed database decreases with the increase of data scale, the Greenplum distributed database was taken as the research object, and a cost-based optimal query plan generation scheme was proposed from the perspective of optimizing the query path. Firstly, an effective cost model was designed to estimate the query cost. The parallel maximum and minimum ant colony algorithm was then used to search the join order with the minimum query cost, i.e. the optimal join order. Finally, the optimal query plan was obtained based on the Greenplum database's default optimal choice for different operations in the query plan. Multiple experiments were carried out on the self-generated data set and TPC-H (Transaction Processing Performance Council Benchmark H) standard data set using the proposed scheme. The experimental results show that the proposed optimization scheme can effectively search out the optimal solution and obtain the optimal query plan, so as to improve the query efficiency of Greenplum database.

Keywords: distributed database; Greenplum database; optimal query plan; cost model; optimal join order

0 引言

在如今大数据时代的背景下，为了满足大规模数据处理及分析的需求，分布式数据库应运而生，并逐渐取代集中式数据库成为当今主流的数据分析系统。但同时，大规模数据的增长降低了分布式数据库的查询效率，如何提升查询效率是分布式数据库领域中的热点和难点。近年来，有不少学者做了寻求查询优化方案的研究，其中，对数据库系统架构和数据处理方式的创新是优化分布式数据库的重要途径。无共享架构（Shared-nothing architecture）[1]是在分布式数据库优化过程中产生的一种集群架构。BigTable[2]、Google File System[3]、MapReduce[4]等都是典型的大规模并行处理（Massive Parallel Processing, MPP）架构。它们的成功推动了无共享架构集群的发展。Greenplum[5]是目前最先进的分布式开源数据库技术之一，其高度并行性的优势将推动并促进基于 MPP 架构的分布式数据库的发展与流行[6]。

收稿日期: 2017-07-31； **修回日期**:2017-09-08。**录用日期**:2017-09-19。
基金项目:国家自然科学基金资助项目（61503289）。
作者简介: 邹承明(1975—)，男，广东徐闻人，教授，博士，CCF 会员（30617M），主要研究方向：计算机视觉、嵌入式系统、软件理论与方法；
谢义(1991—)，男，湖南邵东人，硕士研究生，主要研究方向：软件定义与数据迁移；
吴佩(1993—)，女，湖北武汉人，硕士研究生，主要研究方向：图形图像处理。

查询操作是分布式数据库对用户提供的最基本服务。本文将从最优查询计划生成方法的角度来对 Greenplum 数据库的查询操作进行优化。最优查询计划的生成问题是一个最优解的求解问题。对于最优解的求解，有两种典型的方法：基于启发式的方法和随机搜索方法。动态规划是典型的基于启发式的方法。文献[7-8]都提出了利用动态规划解决优化问题，并将其应用到查询优化中，这种方法虽然能得到最优解，但随着连接关系数的不断增大，搜索效率会变得非常低，动态规划算法不太适合大规模数据场景下的复杂的多表连接查询的优化问题。遗传算法作为一种有效的随机搜索算法，随着问题规模的增大，其优势会比较明显[9]。文献[10]将遗传算法应用于分布式数据库查询计划生成，与动态规划相比，这种方法的搜索效率更高，但是该方法不一定能够保证获得最优解，而且该方法的好坏依赖于种群的初始化、变异交叉操作的选择以及适应度函数的选择。蚁群算法也是一种有效的随机搜索算法。文献[11]提出了一种基于蚁群算法的方法来减少查询过程中连接操作产生的代价，从而获得一个关系数据库复杂查询的最优查询计划，但是该方法的代价模型中只考虑了连接操作产生的代价，忽略了分布式数据库节点之间的数据传输代价。文献[12]提出了最大最小蚁群算法，这种改进的蚁群算法进一步解决了算法过早收敛的问题，求解效率高。经过对上述文献中的算法的对比，本文在文献[12]提出的算法基础上，提出了一种并行最大最小蚁群算法来搜索最优连接顺序。

基于以上研究，为了提升 Greenplum 数据库的查询效率，本文提出了一种有效的基于代价的最优查询计划生成方法，并通过实验验证了其有效性。

1 代价模型

基于代价的查询优化是 MPP 数据库的核心技术。Greenplum 数据库的查询优化器就是以代价作为目标，搜索具有最小代价的查询计划[13]。这种基于代价的查询优化首先根据一个查询语句的逻辑生成树，预估每个查询策略的代价，然后从中选择代价最小的路径作为最终的物理执行计划。因此，代价预估是基于代价的查询优化中的一个重要问题。本文通过对分布式数据库查询代价的组成进行分析，设计了一个有效的查询代价模型。通过这个代价模型，在任务实际执行之前，能够预估整个任务完成的总代价。

一般情况下，集中式数据库系统的总查询代价公式为 $C_{total}=C_{CPU}+C_{I/O}$，但是，分布式数据库是由节点之间互相协作完成整个查询任务，节点之间的交互同时会产生通信代价，所以，分布式数据库的总查询代价公式可表示为 $C_{total}=C_{CPU}+C_{I/O}+C_{trans}$，其中，$C_{trans}$ 表示为节点间的通信代价。通信代价的估算可以通过公式（1）来实现，其中 X 表示需要传输的数据量，C_0 和 C_1 是两个常量系数。

$$C_{trans}(X)=C_0+C_1*X \tag{1}$$

1.1 相关参数

分布式数据库系统中一般都会存储着数据表的统计信息，这些统计信息可以用来对代价进行估计。而这类信息通常是存储在数据库的数据字典中，供查询优化器来使用[6]。从 Greenplum 数据库的数据字典中，可以获得代价估计所需的统计信息。Greenplum 数据库对代价的估计忽略了数据传输的代价，因此，本文将数据传输的代价考虑进来，提出了一种新的代价模型。该模型需要从数据字典中获取的参数主要是 $|R|$ 和 $V(a,R)$。其中，$|R|$ 表示关系 R 中的元组数，$V(a,R)$ 表示关系 R 中的属性 a 所具有的不同值的数目。

1.2 代价估算

假设给定一个连接 $S=R_1\ joinR_2$，R_1 和 R_2 是两个要连接的关系表。从数据传输代价 $Cost_{trans}(S)$ 和连接产生的数据量 $Cost_{join}(S)$ 两个方面来对该连接进行代价估算，如公式（2）所示。

$$Cost_{total}(S)=Cost_{trans}(S)+Cost_{join}(S) \tag{2}$$

1）数据传输代价：影响数据传输代价的最主要因素是进行连接的关系表的规模大小。但是，在网络传输速度很快的情况下，数据传输的代价相对 CPU 对数据进行处理的代价影响较小。为了统一计算连接的代价，本文为数据传输代价设置一个权值 ω，数据传输代价如公式（3）所示。

$$Cost_{trans}(S)=w*(|R_1|+|R_2|) \tag{3}$$

2）连接产生的数据量：某一个连接产生的数据量越大，那么该连接产生的代价就越大，针对该连接的下一个连接的代价也就更大。因此，本文将连接产生的数据量作为连接的代价。对于连接 $S=R_1\ joinR_2$，连接中属性所具有的不同值的数目可通过公式（4）计算。

$$V(a,S)=\begin{cases}V(a,R_1) & a\in R_1-R_2\\ V(a,R_2) & a\in R_2-R_1\\ \max(V(a,R_1),V(a,R_2)) & a\in R_1\cap R_2\end{cases} \tag{4}$$

设 $Attr$ 为关系表 R_1 和 R_2 的公共属性集合，连接产生的数据量可以用公式（5）来表示。

$$Cost_{join}(S)=\frac{|R_1||R_2|}{\prod_{i=1}^{n-1}\max\left(V(Attr_i,R_1),V(Attr_i,R_2)\right)} \tag{5}$$

在分布式数据库中，一个多表连接查询语句的查询执行计划都是用一颗连接树来表示的。假设用连接树的内节点 $S_i(i=1,\ 2,...,n-1)$ 来表示中间连接关系。那么该连接树内

所有内节点 Si的代价总和就可以表示某一个查询计划的总代价。因此，可以用公式（6）来表示一个多表连接查询执行计划的代价模型。

$$Cost_{total}(S)=\sum_{i=1}^{n-1}(Cost_{join}(S_i)+Cost_{trans}(S_i)) \quad (6)$$

2 并行最大最小蚁群算法

Greenplum 数据库中的复杂查询基本上都会涉及到跨表查询。跨表查询的一个最基本操作就是连接，而连接操作也是分布式数据库中一个关键的操作。在连接操作中，内连接满足交换律，而外连接和子查询都可通过一定方式转换为内连接[14]。通常情况下，两个关系表之间的连接操作是最耗时的操作之一[15]。因此，涉及到连接操作的处理代价是影响复杂查询语句代价的主要因素。这种连接代价主要取决于给定的查询语句中所涉及到的关系表的连接顺序。假设数据库中有三个关系表 R_1,R_2 和 R_3,这三个表所有可能的连接顺序有：$(R_1R_2)R_3$，$R_1(R_2R_3)$和$(R_1R_3)R_2$。不同的连接顺序会产生不同的查询计划代价，所以搜索最优的连接顺序对提升查询效率起着重要作用。因此，对连接顺序进行优化是分布式数据库多表连接查询的一个研究重点。一般情况下，影响连接顺序优化的因素有以下三个：搜索空间、代价模型、搜索策略。本文在进行连接顺序优化过程中，采用第 1 节中设计的代价模型。对于搜索策略，本文实现了一种并行化最大最小蚁群算法来解决最优解搜索问题。

将蚁群算法应用到多表连接顺序优化的问题中时，要注意在每次选中连接表进行连接之后，对表的规模和连接进行重新计算。下面将详细介绍并行最大最小蚁群算法运用到连接顺序优化问题中的一般过程，包括：信息初始化、转移概率计算、信息素更新、表规模更新以及并行化处理。

1）信息初始化。对各路径信息素进行初始值设定。设 φ 表示初始信息素，令 $\varphi=C$，其中为 C 常数。

2）转移概率。对第 t 代循环，蚂蚁个体 A_{ij} 从节点 i 转移到 j 的概率 $P_{ij}^{k}(t)$ 的可用公式（7）计算。

$$P_{ij}^{k}(t)=\begin{cases}\dfrac{j_{ij}^{a}(t)h_{ij}^{b}(t)}{\sum j_{ij}^{a}(t)h_{ij}^{b}(t)}, & i,j\in W^{k}\\ 0, & \text{其他}\end{cases} \quad (7)$$

$\varphi_{ij}(t)$表示边(i,j)的信息素强度，α 表示轨迹的相对重要性，β 为期望的相对重要性。W^k 表示一个关系表的集合，这个集合中的元素是所有还未被蚂蚁 k 选择的所有关系表。$\eta_{ij}(t)$表示关系表 i 转移到关系表 j 的期望程度，如公式（8）所示，其中 $Cost_{ij}$ 表示两个关系表连接的代价。

$$h_{ij}=1/Cost_{ij} \quad (8)$$

3）信息素更新。最大最小蚁群算法对传统蚁群算法的改进之处，就是改进信息素更新的方式。最大最小蚁群算法分别采用局部信息素和全局信息素来进行信息素的更新。其中，局部信息素可通过公式（9）来进行更新，对每只蚂蚁，只进行一次局部信息素的更新。

$$j_{ij}(t+1)=dj_{ij}(t)+(1-d)\Delta j_{ij}(t) \quad (9)$$

在进行路径搜索时，依照公式（7）计算出的转移概率，蚂蚁会根据概率大小在可选集中选择出下一步要连接的关系表。$\Delta\varphi_{ij}(t)$模拟初始时刻连接边(i,j)的信息素量，其中 δ 表示蒸发因子，它是一个 0 到 1 的常数。设置 δ 的目的是：防止蚂蚁由于过早收敛而陷入局部最优解。

最大最小蚁群算法在所有蚂蚁完成连接之后，会搜索出一条当前代价最小的连接顺序，经过这条连接顺序的蚂蚁个体被选择成为最佳蚂蚁。算法规定只有这些被选出的最佳蚂蚁可以在经过的路径上留下信息素。也就是说，最大最小蚁群算法对传统蚁群算法的一个改进之处是增强最优解的信息素。同时，最大最小蚁群算法还会削减选出的最差路径上的信息素强度。这个削减操作是为了保证蚂蚁更快地集中到最优路径周围。最大最小蚁群算法的全局信息素更新方式可由公式（10）至（13）表示。

$$j_{ij}(t+n)=dj_{ij}(t)+(1-d)\Delta j_{ij}^{best}(t) \quad (10)$$

$$j_{ij}(t+n)=dj_{ij}(t)+(1-d)\Delta j_{ij}^{worst}(t) \quad (11)$$

$$\Delta j_{ij}^{best}(t)=1/L_{best} \quad (12)$$

$$\Delta j_{ij}^{worst}(t)=1/L_{worst} \quad (13)$$

$\varphi_{ij}(t+n)$表示的是，在所有蚂蚁完成所有关系表的连接后，得到的最优连接顺序中边(i,j)上的信息素。L_{best} 代表此次迭代过程中的最优连接顺序的代价； L_{worst} 表示此次迭代过程中的最差连接顺序的代价。

4）表规模更新。利用蚁群算法处理多表连接顺序优化的过程中，不同于传统旅行商问题的一点是：每次连接时，可选集中会删除被连接的表，同时，被选中的表的规模会发生变化。将 C_{AB} 表示为表 A 连接表 B 之后的表 B 的规模。即：$|B|=C_{AB}$。表 B 中的属性会与表 A 中的属性集合并，表 B 的属性规模可由公式（14）计算，其中 $Attr$ 表示 A 与 B 的公共属性集合。

$$V(Attr_i,B)=\begin{cases}\max(V(Attr_i,A),V(Attr_i,B)), & Attr_i\in Attr\\ V(Attr_i,A)+V(Attr_i,B), & Attr_i\notin Attr\end{cases} \quad (14)$$

5）将最大最小蚁群算法运用到大规模数据表的多表连接查询优化问题中时，算法的搜索时间比较长。针对这一缺陷，本文结合分布式数据库并行性的特点对最大最小蚁群算法进行并行化处理，以达到更快获得最优解的目的。最大最小蚁群算法的并行一般有两种方式：即细粒度的蚂蚁并行方式和

粗粒度的蚁群并行方式。前者以个体为并行单位，可通过消息传递和分布式共享内存的编程模型，以主从模式将蚂蚁个体分配给不同的处理单元来实现并行化，但是这种并行方式会带来处理单元之间频繁的通信消耗；后者以种群为并行单位，也可以通过消息传递和共享变量两种并行编程模型来实现，该方式将蚂蚁种群分配给不同的处理单元来计算，减少了处理单元之间的消息传递。

本文的并行化是通过粗粒度的种群并行方式，采用基于共享变量模型的 Pthread 库，将含有相同数量的蚁群分配给多个线程并行处理（设置蚁群数量相等是为了均衡处理单元的运算负载），以提高搜索效率，从而提升 Greenplum 数据库的查询性能。并行最大最小蚁群算法相对于串行最大最小蚁群算法的区别在于，每个蚁群的蚂蚁在经过局部信息素更新之后，会影响到所有蚁群的蚂蚁的前进路线，因为信息素矩阵是被所有蚂蚁共享的，但是局部信息素的更新会较频繁的访问信息素矩阵，导致同步操作增加而影响算法的运行效率，为此，本文对每只蚂蚁进行限制，只进行一次局部信息素的更新。

最大最小蚁群算法并行化的过程如算法 1 所示。首先根据并行度同时运行数量相等的 k 只蚂蚁的种群的迭代过程。对每一个种群，按照最大最小蚁群算法的搜索过程得到最优路径。将所有种群的路径提交给主线程之后，选择更优的路径，并更新全局信息素。迭代以上过程，直到得到最终的最优路径。

算法 1　并行最大最小蚁群算法

Input:　n 个表及其规模大小

Output: 最优连接顺序

1) Initiation：并行度 P，蚂蚁数量 K，α，β，全局迭代次数 $N1$，局部迭代次数 $N2$
2) for i 1 to $N1$　//全局迭代
3) do parallel　//并行化处理
4) for j 1 to $N2$　//局部迭代
5) for k 1 to K　//选择拥有 k 只蚂蚁的种群进行迭代
6) 初始化可选集 W^k，信息素浓度 φ；
7) while $W^k \neq \varnothing$ do
8) 随机选取 W^k 中的一个表 A 作为起点，将表 A 从 W^k 中删除，更新 W^k；
9) 计算转移概率并根据结果选取一个表 B 与表 A 连接；
10) 修改被选中的表 B 的规模，$|B|=C_{AB}$；局部更新信息素浓度；
11) end while
12) end for
13) end for
14) 提交最优路径；
15) end do
16) 修改全局最优路径，全局更新信息素浓度；
17) end for

3　实验结果及分析

实验环境是 Greenplum 数据库集群，该集群有 4 个节点：一个主节点，三个子节点。每个节点的实验平台均采用 2.5GHZ 双核处理器，8GB 内存，CentOS 7 操作系统，Greenplum 数据库的版本号是 greenplum-db-gpdb-14b78f4。为了满足对比实验的公平性原则，基于 MySQL 数据库与 PostgreSQL 数据库的实验都是在 8 核的处理器上完成。

3.1　自主生成数据集实验

为了验证代价模型的有效性，本文利用 Greenplum 数据库集群做了如下实验。首先在数据库中生成 5 张关系表 A、B、C、D 和 E，每张表的元组数分别为 50000、100000、150000、200000 和 250000。5 张生成表的规模及属性如表 1 所示。此外，本文通过多次实验，得到并行最大最小蚁群算法的最优参数选择，其中蚂蚁数量设置为 30 只，局部迭代次数和全局迭代次数分别设置为 20 次和 30 次，并行度设置为 2，代价模型中的权值设置为 0.1。

表1　自主生成的表的规模及属性

Tab. 1　Size and attributes of the self-generated tables

表 A	表 B	表 C	表 D	表 E
|A|=50000	|B|=100000	|C|=150000	|D|=200000	|E|=250000
V(a,A)=10000	V(a,B)=30000		V(a,D)=100000	V(a,E)=140000
V(b,A)=15000		V(b,C)=80000	V(b,D)=120000	V(b,E)=150000
V(c,A)=10000	V(c,B)=30000	V(c,C)=50000		
	V(d,B)=80000	V(d,C)=120000	V(d,D)=175000	V(d,E)=200000
V(e,A)=20000	V(e,B)=50000	V(e,C)=80000	V(e,D)=120000	V(e,E)=170000

初始的连接顺序按从左至右为((AB)C)D)E)，获取该连接的执行时间，然后利用并行最大最小蚁群算法找出代价最小的连接顺序，即(((A(ED))C)B)，输出这种方式的执行时间并且与其他连接方式的执行时间进行对比。表 2 中列出了其中 5 种连接方式的实际完成时间。对每种连接方式分别进行 10 次实验，取平均值作为该连接顺序进行连接的时间。从实验结果可以看出，搜索出的最优连接顺序完成连接所用的时间最少，这说明本文设计的代价模型可以比较准确的衡量不同连接顺序的代价，证明了该代价模型以及并行最大最小蚁群算法的有效性。

另一方面，为了验证并行最大最小蚁群算法的寻优能力，本文分别利用动态规划、最大最小蚁群算法、遗传算法以及并行最大最小蚁群算法来进行了多组对比实验。图 1 展示了上述四种算法获得最优解的时间对比，当连接关系数为 5 和 10 时，四种算法的寻优效率相差不大；随着关系数的增加，遗传算法和并行最大最小蚁群算法的效率相对其他两种算法来说更高；当连接关系数增加到 25 时，并行最大最小蚁群算法的搜索效率相对遗传算法来说更高，且随着关系数的增多，这种优势更加明显。从图中可以看出，并行最大最小蚁群算法在的寻优能力比 Greenplum 数据库定义的动态规划和遗传算法的效率更高。

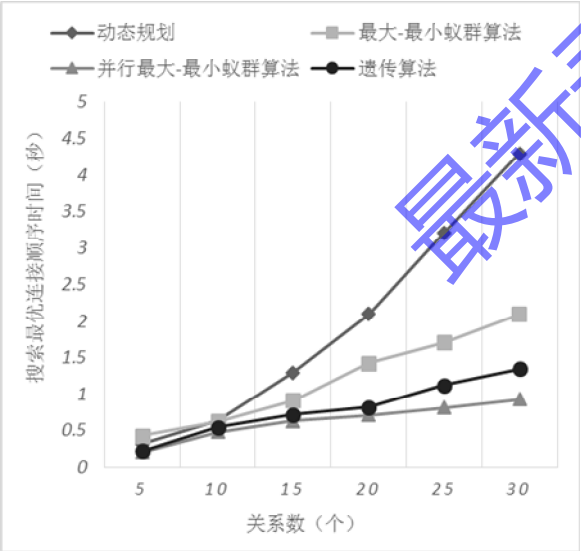

图1 四种算法的搜索最优连接顺序时间比较

Fig. 1 Comparison of search time for optimal join order of four algorithms

表2 连接操作的执行时间

Tab. 2 Execution time of join operation

连接顺序	时间/ms	连接顺序	时间/ms
((((AB)C)D)E)	41.58	(A(B(C(ED))))	43.16
((A((BC)D))E)	42.36	(((A(ED))C)B)	38.01
((A(B(CD)))E)	40.89		

3.2 标准数据集实验

为了验证基于代价模型的并行最大最小蚁群算法对 Greenplum 数据库性能的优化，本文在 Greenplum 数据库集群上模拟了复杂查询。本节实验的数据都是通过 TPC-H 工具得到的，其中测试数据量的统计如表 3 所示。

表3 各测试表的数据量统计

Tab. 3 Number of data in each test table

表名	数据条数	表名	数据条数
CUSTOMER	150000	PART	200000
LINEITEM	6001215	PARTSUPP	800000
NATION	25	REGION	5
ORDERS	1500000	SUPPLIER	10000

TPC-H 是一套针对数据库决策支持能力的测试基准，用来测试数据库系统复杂查询的响应时间，其中定义了一个数据库模型。数据库模型包括 8 张数据表：CUSTOMER、LINEITEM、NATION、ORDERS、PART、PARTSUPP、REGION 和 SUPPLIER。针对这 8 张数据表，提供了 22 条复杂的查询语句。本文通过执行这 22 条复杂查询语句的时间长短来分析查询效率，实验结果展示了前 10 条语句的查询时间。表 4 给出了 Greenplum、MySQL、PostgreSQL 这三种数据库处理查询语句的时间对比。根据表 4 中执行时间的统计，可以看出 MySQL 数据库对查询语句的处理时间最长。而 PostgreSQL 相对 MySQL 的优势则体现在法的性能上，其采用的 Hash join 可以提高系统查询的效率。但是，将 PostgreSQL 与 Greenplum 进行比较时，对于 *S3*、*S7* 和 *S8* 这三条查询语句，PostgreSQL 的处理时间更短。为了更精确的对 PostgreSQL 与 Greenplum 的性能进行对比分析，将 8 张数据表的规模放大 10 倍之后再次对这两个数据库进行实验。其中，前 10 条查询语句的执行结果如表 5 所示。从表中结果可以看出，随着数据量的增大，PostgreSQL 的查询时间显著增加，性能下降比较明显。但是，对 *S7* 这条查询语句，PostgreSQL 的执行时间仍然比 Greenplum 的更短。这是因为，对少数一些查询语句，Greenplum 的重分布会消耗大量时间。同时，对比实验的数据量还不足以模拟 OLAP 场景，不能充分体现 Greenplum 对大数据处理的优势。

表 6 列出了 Greenplum 数据库优化前和优化后的查询语句执行时间对比。优化前后采用的表的规模是表 1 中的规模的 10 倍。从表 6 中可以看出，优化后的 Greenplum 数据库的平均查询执行时间较优化前有所减少，其查询效率更高。而优化后的平均查询执行时间减少幅度不大的原因，是由于本文实验集群中每个子节点配置的仅是双核 CPU，虽然采用 CPU 内核调度可以提升 CPU 内核的利用率，但是，调度过程本身也会消耗时间，更主要的是 Greenplum 数据库的查询过程包括主节点将并行最大最小蚁群算法生成的最优查询计划分发给各子节点，分发过程涉及到广播或重分布引起的子

节点之间的数据传输，在数据规模不大的情况下，数据传输时间占据查询执行时间的比重不可忽略，甚至对执行时间的影响较大。在内核数量少且查询语句与数据量不大的情况下，优化后的 Greenplum 数据库查询性能提升不明显，但较 Greenplum 优化前的查询性能有所提高。

表4 三个数据库的 SQL 语句执行时间对比（单位：秒）

Tab. 4 Comparison of execution time for SQL statements of three databases (unit: second)

QL 语句	MySQL	PostgreSQL	Greenplum
S1	12.66	12.94	4.01
S2	3.27	0.68	0.5
S3	5.06	1.3	1.35
S4	0.01	0.55	0.11
S5	27.26	0.69	0.19
S6	2.5	0.81	0.01
S7	10.79	1.79	6.06
S8	39.78	0.61	1.46
S9	>12h	7.12	4
S10	4.74	2.18	0.14

表5

表6 两个数据库的 SQL 语句执行时间对比（单位：秒）

Tab. 5 Comparison of execution time for SQL statements of two databases (s)

SQL 语句	PostgreSQL	Greenplum	SQL 语句	PostgreSQL	Greenplum
S1	130.61	36.68	S6	11.08	0.01
S2	17.08	3.1	S7	42.96	80.12
S3	117.83	14.39	S8	45.13	6.61
S4	6.81	0.11	S9	118.36	49.72
S5	114.46	0.2	S10	40.51	0.16

表7 Greenplum 优化前后的 SQL 语句执行时间对比（秒）

Tab. 6 Comparison of execution time for SQL statements of Greenplum before and after optimization (s)

SQL 语句	优化前	优化后	SQL 语句	优化前	优化后
S1	36.68	35.79	S6	0.01	0.01
S2	3.10	3.41	S7	80.12	78.83
S3	14.39	13.98	S8	6.61	6.59
S4	0.11	0.09	S9	49.72	48.95
S5	0.20	0.17	S10	0.16	0.14

4 结语

为了解决分布式数据库数据规模增大而导致的查询效率降低的问题，本文对最优查询计划生成问题进行研究，提出了一种最优查询计划生成方法。连接顺序执行时间、算法寻优能力、查询语句执行时间的对比实验表明，考虑了传输代价的代价模型能较准确的估计不同连接顺序的代价，且基于该代价模型的并行最大最小蚁群算法在四种算法中拥有最好的寻优能力。由于实验采用的查询语句、表规模等数量上的有限性，优化后的 Greenplum 数据库在查询性能提升方面不是很明显。在最优查询计划生成之后，为进一步提升 Greenplum 数据库的查询性能，对各节点之间的任务调度进行优化是下一步的研究方向。

参考文献

[1] ZENG F S. Research and improvement of database storage method [J]. Applied Mechanics & Materials, 2014, 608-609: 641-645.

[2] CHANG F, DEAN J, GHEMAWAT S, et al. Bigtable: a distributed storage system for structured data [J]. ACM Transactions on Computer Systems, 2008, 26(2): 4.

[3] GHEMAWAT S, GOBIOFF H, LEUNG S T. The Google file system [C]//SOSP 2003: Proceedings of the 19th ACM Symposium on Operating Systems Principles. New York:ACM, 2003, 37: 29-43.

[4] DEAN J, GHEMAWAT S. MapReduce: simplified data processing on large clusters [J]. Communications of the ACM, 2008, 51(1): 107-113.

[5] GOLLAPUDI S. Getting started with Greenplum for big data analytics [M]. Birmingham: Packt Publishing, 2013: 24.

[6] ANTOVA L, EL-HELW A, SOLIMAN M A, et al. Optimizing queries over partitioned tables in MPP systems [J]. ACM, 2014, 33 (37-38): 373-384.

[7] RODRIGUES E R, NAVAUX P O A, PANETTA J, et al. A comparative analysis of load balancing algorithms applied to a weather forecast model [C]// SBAC-PAD 2010: Proceedings of the 22nd International Symposium on Computer Architecture and High Performance Computing. Piscataway, NJ:IEEE, 2010: 71-78.

[8] KOSSMANN D, STOCKER K. Iterative dynamic programming: a new class of query optimization algorithms [J]. ACM Transactions on Database Systems, 2000, 25(1): 43-82.

[9] ILAVARASAN E, THAMBIDURAI P. Genetic algorithm for task scheduling on distributed heterogeneous computing system [J]. International Review on Computers & Software, 2015, 1(3): 233.

[10] KUMAR T V V, SINGH V, VERMA A K. Distributed query processing plans generation using genetic algorithm [J]. International Journal of Computer Theory and Engineering, 2011, 3(1): 38.

[11] HANAFY H A, GADALLAH A M. Ant Colony-Based Approach for Query Optimization [C]//DMBD 2016: Proceedings of the 2016 International Conference on Data Mining and Big Data. Berlin:Springer International Publishing, 2016: 425-433.

[12] HOOS H H, STUETZLE T. Special issue on stochastic search algorithms-Preface [J]. 2007, 156(1): 1-4.

[13] SOLIMAN M A, ANTOVA L, RAGHAVAN V, et al. Orca: a modular query optimizer architecture for big data [C]// SIGMOD 2014: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. New York:ACM, 2014: 337-348.

[14] GIRI A K, KUMAR R. Distributed query processing plan generation using iterative improvement and simulated annealing [C]// IACC 2013: Proceedings of the 3rd International Advance Computing Conference. Piscataway, NJ:IEEE, 2013: 757-762.

[15] MAHMOUD F, SHABAN S, ABD EL-NABY H: A proposed query optimizer based on genetic algorithms [J]. Egypt. Computer Science, 2010, 37(1): 1-22.

This work is partially supported by the National Natural Science Foundation of China (61503289).

ZOU Chengming, born in 1975, Ph. D., professor. His research interests include computer vision, embedded system, and software theory and methods.

XIE Yi, born in 1991, M. S. candidate. His research interests include software definition and data migration.

WU Pei, born in 1993, M. S. candidate. Her research interests include graphic image processing.